FreeBSD Container Images

BY DOUG RABSON

OCI container engines such as containerd or podman need images. A container image is a read-only directory tree which typically contains an application with supporting files and libraries. Running this image on a container engine makes a writable clone of the image and executes the application in some kind of isolation environment such as a jail.

Images are distributed via registries which store the image data and provide a simple REST API to access images and their metadata. The registry APIs, image formats and metadata are standardised by the Open Container Initiative which largely replaces earlier docker formats.

Running this image on a container engine makes a writable clone of the image and executes the application in some kind of isolation environment such as a jail.

OCI Images

Images are represented as a sequence of layers, each of which is stored as a compressed tar file. To unpack the image, we start with an empty directory and then unpack each layer in sequence, allowing later layers to add files or change files from an earlier layer. Typically, the result of this process is cached by the container engine.

In addition to the layer data, two additional metadata objects are used. The manifest lists the layers and can contain annotations to describe the image. The image config describes the target operating system and architecture and allows a default command to be used for running the image.

All of this is stored in a 'content addressable' structure where the hash of a component is used to name it. For example, a small base image I use for statically linked applications looks like this:

```
$ ls -lR
total 6
drwxr-xr-x  3 root  dfr    3 Sep  8 10:36 blobs
-rw-r--r--  1 root  dfr  275 Sep  8 10:36 index.json
-rw-r--r--  1 root  dfr   31 Sep  8 10:36 oci-layout

./blobs:
total 25
drwxr-xr-x  2 root  dfr  6 Sep  8 10:36 sha256

./blobs/sha256:
total 950
-rw-r--r--  1 root  dfr    1143 Sep  8 10:36
190e4f8bf39f4cc03bf0f723607e58ac40e916a1c15bd212486b6bb0a8c30676
-rw-r--r--  1 root  dfr     496 Sep  8 10:36
5657eb844c0c0142aa262395125099ae065e791157eaa1e1d9f5516531f4fe30
-rw-r--r--  1 root  dfr   34916 Sep  8 10:36
5af368a2a6078dc912135caed94a6375229a5a952355f5fea60dad1daf516f78
-rw-r--r--  1 root  dfr  911102 Sep  8 10:36
fdb4ee0a131a70df2aae5c022b677c5afbacb5ec19aa24480f9b9f5e8f30fd18
```

All the metadata files in this bundle are in json format as described here. The top-level index.json file links to the manifest using its hash:

```
$ cat index.json | jq
{
  "schemaVersion": 2,
  "manifests": [
    {
      "mediaType": "application/vnd.oci.image.manifest.v1+json",
      "digest": "sha256:190e4f8bf39f4cc03bf0f723607e58ac40e916a1c15bd212486b6bb0a8c30676",
      …
    }
  ]
}
```

This manifest describes two data layers, one with just the FreeBSD standard directory structure and one containing minimal support files such as /etc/passwd and ssl certificates. It also links to the config which has the target operating system and architecture.

Using a content-addressable format like this makes it easier to share storage space and reduce the amount of data downloaded when using multiple images derived from the same base.

The OCI image specification also allows for multi-architecture images which are just lists of manifests:

```
{
    "schemaVersion": 2,
    "mediaType": "application/vnd.docker.distribution.manifest.list.v2+json",
    "manifests": [
        {
            "mediaType": "application/vnd.oci.image.manifest.v1+json",
            "size": 1116,
            "digest":
"sha256:598b927b8ddc9155e6d64f88ef9f9d657067a5204d3d480a1b1484da154e7c4",
            "platform": {
                "architecture": "amd64",
                "os": "freebsd"
            }
        },
        {
            "mediaType": "application/vnd.oci.image.manifest.v1+json",
            "size": 1118,
            "digest":
"sha256:ac732db0f4788d5282a8d16fefbea360d937049749c83891367abd02801b582",
            "platform": {
                "architecture": "arm64",
                "os": "freebsd"
            }
        }
    ]
}
```


FreeBSD Base Images

To make it easier to work with containers on FreeBSD, there is a need for suitable base images. The traditional FreeBSD release process generates a small number of packages intended for installing a fully featured FreeBSD OS on a physical or virtual host. We could use the base.txz package to build our base image but this results in a gigabyte sized image, more than 90% of which is not needed by most applications. Most Linux distributions offer much smaller base images — the official Ubuntu image, for instance, is about 80MB.

Most Linux distributions offer much smaller base images — the official Ubuntu image, for instance, is about 80MB.

Fortunately, the pkgbase project has been working to make a fine-grained package set which subdivides the traditional base.txz tarball into hundreds of much smaller packages. Currently, this consists of many packages for individual libraries and utilities along with two larger packages, FreeBSD-runtime which contains the shell along with a selection of core utilities and FreeBSD-utilities which has a larger set of commonly used utilities.

Early on, I created a "minimal" image using pkgbase which included FreeBSD-runtime, plus SSL certificates and pkg. This is about 80MB and contains enough functionality for sim-

ple shell scripts as well as the ability to install packages. This compares favourably with similar Linux images although it doesn't come close to the busybox-based alpine image which is just 7.5MB.

Since then, I made a small family of images, partly inspired by the distroless project:

- "static" which contains just SSL certificates and timezone data. This can be used as a basis for statically linked applications.
- "base" which extends "static" by adding a selection of shared libraries to support a wide variety of dynamically linked applications.
- "minimal" which adds the FreeBSD-runtime package and package management as before
- "small" which adds FreeBSD-utilities for broader support of shell-based applications.

To support a variety of FreeBSD versions, I embed the version into the image name, e.g., "freebsd13.2-minimal:latest" includes packages from the most recent version of the releng/13.2 branch while "freebsd13-minimal:latest" is built from stable/13. I build all these images with support for amd64 and arm64 architectures and the container engine will automatically select the correct image from the manifest list.

Security

It is important that container images can be verified that they have a trusted origin and have not been tampered with while they are being transferred to the container engine.

An image's manifest typically contains the SHA256 hashes of the image's data layers as well as the hash of the corresponding image config. This means that the hash of the manifest can be used to uniquely identify the image. This can be used to verify the image, e.g., by listing trusted image hashes in a trustable location.

Alternatively, the hash can be used to create a signature which can prove that the image is trusted by the owner of some public key.

Alternatively, the hash can be used to create a signature which can prove that the image is trusted by the owner of some public key. Two common mechanisms are in use for this — the sigstore facility used by podman uses PGP to create an image signature and provides a mechanism to associate a set of images with a signature store which can either be a local directory or a trusted website. This can be used when an image is pulled to verify that it matches the signature. An alternative to sigstore is cosign which stores the signatures alongside the images in the image repository.

Limitations and Future Work

While these images are useful, they contain a fairly arbitrary choice of which packages are installed. Initially, I included support for the alpha.pkgbase.live package repository which simplified extending an image by installing extra packages. Unfortunately, this project lost its funding and for a while, there was no publicly available pkgbase repository. Thankfully, this has been resolved with pkgbase packages available from the standard FreeBSD package repository.

The current mechanism for building images uses pkg to install pkgbase packages into image layers. This is convenient and keeps a record of what was installed into the image. Unfortunately, the pkg metadata is stored in a sqlite database and this does not support

reproducible builds. The sqlite database includes the timestamp a package was installed — this can be overridden to some suitable constant time but even then, the sqlite database is not reproducible.

A larger issue is credibility — I host these images in my own personal repositories at docker.io and quay.io but from the perspective of potential users, there is no reason to trust that the images are trustworthy. Even though I can build images using packages from the FreeBSD package repository these images are not signed or supported by the FreeBSD project.

In my opinion, this is a significant barrier for potential users of FreeBSD container engines and blocks moving these projects from their current 'experimental' state to something which can be considered for production. This has been confirmed with several recent conversations about supporting FreeBSD as a platform for open source projects which build and use images.

Ideally, as well as hosting pkgbase package sets, the FreeBSD project should build FreeBSD container images, either hosting an image registry or making these images available on a public repository such as docker.io. I plan to prototype additions to the release building infrastructure to integrate container image building into the existing pkgbase framework which may help to move this forward.

DOUG RABSON is a Software Engineer with more than thirty years of experience ranging from 8-bit text adventure games back in the 1980s to tera-byte-per-second distributed long aggregation systems in the 2020s. He has been a FreeBSD project member and committer since 1994 and is currently working on improving FreeBSD support for modern container orchestration systems such as podman and kubernetes.